```c
// Author: Ali Ismail
// Description: My final project is called Heli_Tilt. It is an accelerometer based game where
//              a helicopter must avoid oncoming pillars. The game has 4 options: game play,
//              high scores, set difficulty and calibrate. The movement of the helicopter is
//              dictated by an accelerometer and drawn on a GLCD. This program contains all the driver firmware to
//              to read the accelerometer, drivers to draw on the GLCD, and a game loop that generates
//              levels, updates helicopter movement, and keeps score.

#include <at89c51ed2.h>
#include <mcs51reg.h>
#include <8052.h>
#include <stdio.h>
#include <stdlib.h>
#include <mcs51/8051.h>
#include "init.h"
#include "lcd.h"
#include "i2c.h"
#include "terminal_comm.h"
#include "ADXL345.h"
#include "font5x8.h"

//Defines
#define FORMAT          0x0B
#define RATE            0x0F
#define BOTTOM          0
#define TOP             1
#define LCD_ROWS        8
#define HELI_WIDTH      5
#define BRICK_WIDTH     5
#define PILLAR_WIDTH    7
#define CLEAR_LINE      1
#define MAX_PILLARS     17
#define PILLAR_CONIG    2
#define NOT_SOLID       0
#define SOLID           1
#define MAX_LEVELS      10
#define FINAL_LEVEL     9
#define IMPOSSIBLE      3
#define SET_DIFFICULTY  2
#define BASE_SCORE_ADD  0x00
#define INT_OFFSET      2
#define BASE_10         10
#define SCORE_PAGE      0
#define SCORE_LINE      40
#define SCORE_MASK      0xFF00
#define COLLIDED        1
#define BOUND_HELI_R    123
#define BOUND_HELI_L    2
#define BOUND_HELI_U    1
#define BOUND_HELI_D    6


//Function Declarations
void update_heli();
void generate_level(unsigned int level_top[MAX_PILLARS][PILLAR_CONIG], unsigned int level_bott[MAX_PILLARS][PILLAR_CONIG], unsigned char difficulty, unsigned char *num_pillars, unsigned char *game_delay);
void draw_level(unsigned int level_top[MAX_PILLARS][PILLAR_CONIG], unsigned int level_bott[MAX_PILLARS][PILLAR_CONIG], int start_point, unsigned char num_pillars);
unsigned char detect_collision(unsigned int level_top[MAX_PILLARS][PILLAR_CONIG], unsigned int level_bott[MAX_PILLARS][PILLAR_CONIG], int start_point, unsigned char num_pillars);
void play_game(unsigned char difficulty);
void high_scores();
unsigned int read_score(unsigned char score);
void score_update(unsigned int score);
void write_score(unsigned int score, unsigned char score_addr);

//Global Variables
int heli_page;
unsigned char heli_line;
unsigned int cheat_mode = 0;
unsigned char calibrate_mode = 0;
unsigned char calibrate = 0;

//Set a variable located where CKCON0 register is
//So I can set X2
__sfr __at (0x8F) CKCON0;

_sdcc_external_startup()
{
    //Utilize all of the XRAM
```

```
78      AUXR |= 12;
79      //Utilize X2 mode
80      CKCON0 |= 0x01;
81      return 0;
82  }
83
84  void program_init()
85  {
86      //Initialize timer1, RS232, GLCD, and the Accelerometer
87      timer1_init();
88      RS232_init();
89      lcdinit();
90      ADXL345_init();
91      //Disable interrupts and put accelerometer in bypass
92      ADXL345_disable_ints();
93      ADXL345_enable_bypass();
94
95      //Initialize helicopter location, cheat mode flag
96      //and calibration flags
97      heli_page = 3;
98      heli_line = 3;
99      cheat_mode = 0;
100     calibrate_mode = 0;
101     calibrate = 0;
102
103     //Enable external interrupt
104      IEN0 |= 0x81;      //Enable /INT0 interrupt
105      IT0 = 1;           //Edge Triggered button press
106
107      //Turn the haptic motor OFF
108      MOTOR_OFF;
109 }
110
111 void main(void)
112 {
113     //Variables
114     unsigned char game_choice, difficulty = 0;
115     program_init();
116
117     //Game Loop
118     while(1)
119     {
120         //Display game menu to user and prompt for
121         //a menu option
122         game_choice = game_menu();
123
124         switch(game_choice)
125         {
126             //Play Game
127             case 1: play_game(difficulty);
128                     break;
129             //Set Game Difficulty
130             case 2: difficulty = difficulty_menu();
131                     break;
132             //Display High Scores
133             case 3: high_scores();
134                     break;
135             //Calibrate Accelerometer
136             case 4: calibrate_mode = 1;
137                     accel_screen();
138                     break;
139             default: printf_tiny("EVERYTHING BROKE!\n\r");
140                     break;
141         }
142     }
143
144 }
145 // Function: play_game()
146 // Input   : difficulty level
147 // Descrip : This function contains the game loop where the level is
148 //           drawn and updated, where the helicopter is drawn and updated
149 //           where the score is kept, and where collisions are detected
150 void play_game(unsigned char difficulty)
151 {
152     //Variables
153     static unsigned char credits = 0;
154     unsigned char a_score[100];
155     unsigned int level_up[MAX_PILLARS][PILLAR_CONIG];
156     unsigned int level_bott[MAX_PILLARS][PILLAR_CONIG];
157     int start_point = 128;
158     unsigned int score = 0;
159     unsigned char collision = 0;
160     unsigned char num_pillars, game_delay, i;
161
```

```
162         //Only display credits once
163         if(credits == 0)
164         {
165             creater_screen();
166             credits = 1;
167         }
168         //Display still screen
169         still_screen();
170 
171         //Start helicopter at an initial position
172         GLCD_WriteHeli(HELI,3,3);
173 
174         //Display the start of the level
175         next_level_screen(0);
176 
177         //Turn on the haptic motor
178         MOTOR_ON;
179 
180         //There are 10 levels
181         for(i = 0; i < MAX_LEVELS && !collision; i++)
182         {
183             //Check if the user is in the final level
184             //If they are, draw the impossible level
185             //Otherwise, generate the next level
186             if(i != FINAL_LEVEL)
187                 generate_level(level_up,level_bott, difficulty, &num_pillars, &game_delay);
188             else
189                 generate_level(level_up,level_bott, IMPOSSIBLE, &num_pillars, &game_delay);
190 
191             //Continue the course of the game until there is a collision or the level has ended
192             while(!collision)
193             {
194                 //Draw the pillars
195                 draw_level(level_up, level_bott, start_point, num_pillars);
196                 //Draw the helicopter position
197                 update_heli();
198 
199                 //Check for collision if not in cheat mode
200                 if(!cheat_mode)
201                     collision = detect_collision(level_up, level_bott, start_point, num_pillars
    );
202 
203                 //Introduce game delay to reduce the frames per second, which, will increase
    quality of game experiance
204                 delay_ms(game_delay);
205                 //Increase the gamers score for each pixel survived
206                 score++;
207                 //Convert score to ASCII so it can be printed to GLCD
208                 _itoa(score, a_score, BASE_10);
209                 //Write score to GLCD screen
210                 GLCD_WriteString(a_score, SCORE_PAGE, SCORE_LINE, INVERT);
211                 //Check if the last pillar has left the screen
212                 //Break the loop if it has
213                 if(--start_point == -30 * num_pillars)
214                     break;
215             }
216             //If the gamer completes a level then tell them the level they completed
217             //and the next level they are starting
218             if(!collision && i != FINAL_LEVEL)
219             {
220                 next_level_screen(i + 1);
221                 //Reset starting point of pillars to be drawn to screen
222                 start_point = 128;
223             }
224             //If there is a collision
225             else if(collision)
226             {
227                 //Turn the motor off since they gamer died
228                 MOTOR_OFF;
229                 //Give gamer time to see where they crashed
230                 delay_ms(1500);
231                 //Display to user that the game is over
232                 game_over();
233             }
234             else
235             {
236                 //If user completed the game, they must have cheated
237                 //Call them a cheater
238                 cheater();
239             }
240 
241             //Reset Heli Position
242             heli_page = 3;
243             heli_line = 3;
```

```
244         }
245         //List score as a high score if it is so
246         score_update(score);
247 
248     }
249     // Function: high_scores()
250     // Descrip : This function displays the top 5 high scores of the game onto the GLCD
251     void high_scores()
252     {
253         //Variables
254         unsigned int score = 0;
255         unsigned char a_score[10];
256         unsigned char i;
257         int x = 0;
258 
259         clear_game_screen();
260 
261         //Display each score onto the GLCD
262         for(i = 0; i < 5; i++)
263         {
264             //Read each stored score
265             score = read_score(i + 1);
266             //Converted to ASCII
267             _itoa(score, a_score, 10);
268             //Display score onto the GLCD
269             GLCD_WriteChar(i + 1 + ASCII_OFFSET, i + 1, 45, NORMAL);
270             GLCD_WriteChar('.', i + 1, 50 , NORMAL);
271             GLCD_WriteString(a_score, i + 1, 60, NORMAL);
272         }
273 
274         //Slight delay to account from page transition
275         delay_ms(4000);
276 
277         //Wait until gamer tilts controller to the right to quit this menu option
278         while(x > RIGHT_UP_MIN)
279         {
280             x = ADXL345_read_x();
281         }
282 
283     }
284     // Function: score_update()
285     // Input   : gamers score after losing or beating game
286     // Descrip : This function compares games score to high scores and
287     //           shifts scores accordingly.
288     void score_update(unsigned int score)
289     {
290         //Variables
291         unsigned char i, j;
292         unsigned int score_addr = 0;
293         unsigned int score_val = 0;
294 
295         //Iterate through each score
296         for(i = 1; i < 6; i++)
297         {
298             //Read each score from the EPROM
299             score_val = read_score(i);
300             //If the gamer has a high score
301             if(score > score_val)
302             {
303                 //Shift scores lower than gamer score down
304                 for(j = 4; j >= i; j--)
305                 {
306                     //Calculate the address of where the lower EPROM score should be placed
307                     score_addr = (((j) * INT_OFFSET) + BASE_SCORE_ADD);
308                     //Read the lower EPROM score
309                     score_val = read_score(j);
310                     //Write is to the calculated location
311                     write_score(score_val, score_addr);
312                 }
313                 //Write games high score to the correct location and exit loop
314                 score_addr = (((i - 1) * INT_OFFSET) + BASE_SCORE_ADD);
315                 write_score(score,score_addr);
316                 break;
317             }
318         }
319     }
320     // Function: read_score()
321     // Input   : a score number 1 - 5
322     // Output  : returns a the score value to the prompted score number
323     // Descrip : This function returns the value of the a high score 1 - 5
324     unsigned int read_score(unsigned char score)
325     {
326         //Variables
327         unsigned int score_addr = (((score - 1) * INT_OFFSET) + BASE_SCORE_ADD);
```

```
328         unsigned char i, read_val;
329         unsigned int score_val = 0;
330 
331         //EPROM contains data in bytes
332         //Score contains 2 bytes, thus, 2 bytes are read that represent overall score
333         for(i = 0; i < 2; i++)
334         {
335             //Shift score 8 bits to the left
336             score_val <<= 8;
337             //Read a score byte from EPROM
338             read_val = eebyter(score_addr + i);
339             //Or the read byte to overall score
340             score_val |= eebyter(score_addr + i);
341         }
342         return score_val;
343     }
344     // Function: write_score()
345     // Input    : a gamers score and score addresss
346     // Descrip : This function writes a score to a particular address in the EPROM
347     void write_score(unsigned int score, unsigned char score_addr)
348     {
349         //Variables
350         unsigned char byte_val, i;
351 
352         //Write each byte of the score to the EPROM
353         for(i = 0; i < 2; i++)
354         {
355             //Mask a byte of the score and shift 8 bits to the right
356             byte_val = ((score & SCORE_MASK) >> 8);
357             //Write to the byte to the EPROM
358             eebytew(score_addr + i, byte_val);
359             //Grab the next byte of the score
360             score <<= 8;
361             //Ensures the write finishes
362             delay_ms(5);
363         }
364     }
365     // Function: detect_collision()
366     // Input    : ceiling level pillars and offsets, floor pillars and offset, base position
    for pillars, and number of pillars
367     // Descrip : This function detects a collision of a helicopter into a celling or floor
    pillar
368     unsigned char detect_collision(unsigned int level_top[MAX_PILLARS][PILLAR_CONIG], unsigned
    int level_bott[MAX_PILLARS][PILLAR_CONIG], int start_point, unsigned char num_pillars)
369     {
370         //Variables
371         unsigned int i;
372         int pillar_start_up, pillar_start_bott;
373 
374         //For each pillar that is dictated by the game difficulty
375         for(i = 0; i < num_pillars; i++)
376         {
377             //Calculate the offset of the pillar at the top and bottom of the screen
378             pillar_start_up = start_point + level_top[i][1];
379             pillar_start_bott = start_point + level_bott[i][1];
380 
381             //Check if the helicopter resides on any of the GLCD pages that a ceiling pillar
    occupies
382             if(heli_page <= level_top[i][0])
383             {
384                 //Check if the front column of the helicopter is between the boundry of a pillar
385                 //Excluded self clearing boundry around all moving objects
386                 if((heli_line + HELI_WIDTH - CLEAR_LINE >= pillar_start_up + CLEAR_LINE) && (
    heli_line + HELI_WIDTH - CLEAR_LINE <= pillar_start_up + BRICK_WIDTH))
387                 {
388                     draw_pillar(level_top[i][0], TOP, start_point + level_top[i][1], SOLID);
389                     return COLLIDED;
390                 }
391                 //Check if the back column of the helicopter is between the boundry of a pillar
392                 //Excluded self clearing boundry around all moving objects
393                 else if((heli_line + CLEAR_LINE >= pillar_start_up + CLEAR_LINE) && (heli_line
     + CLEAR_LINE <= pillar_start_up + BRICK_WIDTH))
394                 {
395                     draw_pillar(level_top[i][0], TOP, start_point + level_top[i][1], SOLID);
396                     return COLLIDED;
397                 }
398             }
399             //Check if the helicopter resided on any of the GLCD pages that a floor pillar
    occupies
400             else if(heli_page >= (LCD_ROWS - level_bott[i][0] - 1 ))
401             {
402                 //Check if the front column of the helicopter is between the boundry of a pillar
403                 //Excluded self clearing boundry around all moving objects
404                 if((heli_line + HELI_WIDTH - CLEAR_LINE >= pillar_start_bott + CLEAR_LINE) &&
```

```
    (heli_line + HELI_WIDTH - CLEAR_LINE <= pillar_start_bott + BRICK_WIDTH))
405             {
406                 draw_pillar(level_bott[i][0], BOTTOM, start_point + level_bott[i][1],
    SOLID);
407                 return COLLIDED;
408             }
409             //Check if the back column of the helicopter is between the boundry of a pillar
410             //Excluded self clearing boundry around all moving objects
411             else if((heli_line + CLEAR_LINE >= pillar_start_bott + CLEAR_LINE) && (
    heli_line + CLEAR_LINE <= pillar_start_bott + BRICK_WIDTH))
412             {
413                 draw_pillar(level_bott[i][0], BOTTOM, start_point + level_bott[i][1],
    SOLID);
414                 return COLLIDED;
415             }
416         }
417     }
418     return 0;
419 }
420 // Function: draw_level()
421 // Input   : ceiling level pillars and offsets, floor pillars and offset, base position
    for pillars, and number of pillars
422 // Descrip : This function draws the ceiling and floor pillars based on a base starting
    point
423 void draw_level(unsigned int level_top[MAX_PILLARS][PILLAR_CONIG], unsigned int level_bott[
    MAX_PILLARS][PILLAR_CONIG], int start_point, unsigned char num_pillars)
424 {
425     //Variables
426     unsigned char i;
427     //Draw each pillar
428     for(i = 0; i < num_pillars; i++)
429         {
430             //Draw celing and floor pillars
431             draw_pillar(level_top[i][0], TOP, start_point + level_top[i][1], NOT_SOLID);
432             draw_pillar(level_bott[i][0], BOTTOM, start_point + level_bott[i][1],
    NOT_SOLID);
433         }
434 }
435 // Function: generate_level()
436 // Input   : ceiling level pillars and offsets, floor pillars and offset, base position
    for pillars, game difficulty, number of pillars, and game delay
437 // Descrip : This function generates the orientation and offsets of the ceiling and floor
    pillars. It also sets game setting based on the difficult, such as
438 //           the number of pillars and the game delay
439 void generate_level(unsigned int level_top[MAX_PILLARS][PILLAR_CONIG], unsigned int
    level_bott[MAX_PILLARS][PILLAR_CONIG], unsigned char difficulty, unsigned char *num_pillars
    , unsigned char *game_delay)
440 {
441     //Variables
442     unsigned int i, pillar_len_up, pillar_len_bott;
443     unsigned char min_rand = 0, max_rand = 0, impossible = 0;
444     unsigned int offset_up = 0;
445     unsigned int offset_bott = 15;
446 
447     switch(difficulty)
448     {
449         //Too Easy setting
450         //Pillars can only be a size between 1 - 4 pages, only has 10 pillars (celing and
    floor), and has a game delay of 200/2 ms (X2 mode)
451         case 0: max_rand = 4;
452                 min_rand = 1;
453                 *num_pillars = 5;
454                 *game_delay = 200;
455                 break;
456         //Too Medium setting
457         //Pillars can only be a size between 1 - 5 pages, only has 20 pillars (celing and
    floor), and has a game delay of 100/2 ms (X2 mode)
458         case 1: max_rand = 6;
459                 min_rand = 1;
460                 *num_pillars = 10;
461                 *game_delay = 100;
462                 break;
463         //Smashing cat setting
464         //Pillar are still only between 1 - 5 pages, however, size 5 pillars are more
    likely to occur, 30 pillars are present (ceiling and floor), and has a game delay of 40/2
    ms (X2 mode)
465         case 2: max_rand = 10;
466                 min_rand = 1;
467                 *num_pillars = 15;
468                 *game_delay = 40;
469                 break;
470         //Impossible setting
471         //Draw 3 pillars of size 6 pages
472         case 3: impossible = 1;
```

```
473                 *num_pillars = 3;
474                 *game_delay = 20;
475                 break;
476         default: max_rand = 3;
477                 *num_pillars = 7;
478                 *game_delay = 200;
479                 break;
480     }
481     //For ceiling and floor pillars
482     for(i = 0; i < *num_pillars; i++)
483     {
484         if(!impossible)
485         {
486             //Randomly generate a size for the ceiling and floor pillar
487             pillar_len_up = rand(min_rand,max_rand);
488             pillar_len_bott = rand(min_rand,max_rand);
489             //Ensure pillars are never greater than 5
490             if(pillar_len_up > 5)
491                 pillar_len_up = 5;
492             if(pillar_len_bott > 5)
493                 pillar_len_bott = 5;
494         }
495         //If gamer is on last lever, draw 3 ceiing pillars of size 6
496         else
497         {
498             pillar_len_up = 6;
499             pillar_len_bott = 0;
500         }
501         //Store ceiling and floor sizes and offsets
502         level_top[i][0] = pillar_len_up;
503         level_top[i][1] = offset_up;
504         level_bott[i][0] = pillar_len_bott;
505         level_bott[i][1] = offset_bott;
506         //Increase offset for next pillars so they are nicely spaced out
507         offset_up += 30;
508         offset_bott += 30;
509     }
510 }
511 // Function: update_heli()
512 // Descrip : This function updates the location of the helicopter based on accelerometer
    readings
513 void update_heli()
514 {
515     //Variables
516     int x,y;
517 
518     //Read the x and y axis measurements of the accelerometer
519     y = ADXL345_read_y();
520     x = ADXL345_read_x();
521 
522     //If there is no tilt in the x and y direction, keep the heli in the same spot
523     if(y >= STILL_MIN && y <= STILL_MAX && x >= STILL_MIN && x <= STILL_MAX)
524     {
525         GLCD_WriteHeli(HELI,heli_page,heli_line);
526     }
527     //If there is a tilt to the right, draw the helicopter shifted over to the right
528     else if(y >= STILL_MIN && y <= STILL_MAX && x <= RIGHT_UP_MIN)
529     {
530         //Ensure the helicopter does not cross the boundry of the screen
531         if(++heli_line == BOUND_HELI_R)
532             heli_line = 122;
533         GLCD_WriteHeli(HELI,heli_page,heli_line);
534     }
535     //If there is a tilt to the left, draw the helicopter shifted over to the left
536     else if(y >= STILL_MIN && y <= STILL_MAX && x >= LEFT_DOWN_MIN)
537     {
538         //Ensure the helicopter does not cross the boundry of the screen
539         if(--heli_line == BOUND_HELI_L)
540             heli_line = 3;
541         GLCD_WriteHeli(HELI,heli_page,heli_line);
542     }
543     //If there is a tilt up, draw the helicopter up a page
544     else if (y <= RIGHT_UP_MIN && x >= STILL_MIN && x <= STILL_MAX)
545     {
546         //Ensure the helicopter does not cross the boundry of the screen
547         if(heli_page != BOUND_HELI_U)
548             heli_page--;
549         GLCD_WriteHeli(HELI,heli_page,heli_line);
550         GLCD_WriteHeli(' ',heli_page + 1, heli_line);
551     }
552     //If there is a tilt up and right, draw the helicopter up a page and shifted over to
    the right
553     else if (y <= RIGHT_UP_MIN && x <= RIGHT_UP_MIN)
554     {
```

```
555             //Ensure the helicopter does not cross the boundry of the screen
556             if(heli_page != BOUND_HELI_U)
557                 heli_page--;
558             if(++heli_line == BOUND_HELI_R)
559                 heli_line = 122;
560             GLCD_WriteHeli(HELI,heli_page,heli_line);
561             GLCD_WriteHeli(' ',heli_page + 1, heli_line - 1);
562         }
563         //If there is a tilt up and left, draw the helicopter up a page and shifter over to
    the left
564         else if (y <= RIGHT_UP_MIN && x >= LEFT_DOWN_MIN)
565         {
566             //Ensure the helicopter does not cross the boundry of the screen
567             if(heli_page != BOUND_HELI_U)
568                 heli_page--;
569             if(--heli_line == BOUND_HELI_L)
570                 heli_line = 3;
571             GLCD_WriteHeli(HELI,heli_page,heli_line);
572             GLCD_WriteHeli(' ',heli_page + 1, heli_line + 1);
573         }
574         //If there is a tilt down, draw the helicopter down a page
575         else if(x >= STILL_MIN && x <= STILL_MAX)
576         {
577             //Ensure the helicopter does not cross the boundry of the screen
578             if(heli_page != BOUND_HELI_D)
579                 heli_page++;
580             GLCD_WriteHeli(HELI,heli_page,heli_line);
581             GLCD_WriteHeli(' ',heli_page - 1, heli_line);
582         }
583         //If there is a tilt down and a tilt right, draw the helicopter down a page and
    shifted to the right
584         else if(x <= RIGHT_UP_MIN)
585         {
586             //Ensure the helicopter does not cross the boundry of the screen
587             if(heli_page != BOUND_HELI_D)
588                 heli_page++;
589             if(++heli_line == BOUND_HELI_R)
590                 heli_line = 122;
591             GLCD_WriteHeli(HELI,heli_page,heli_line);
592             GLCD_WriteHeli(' ',heli_page - 1, heli_line);
593         }
594         //If there is a tilt down and a tilt to the left, draw the helicopter down a page and
    shifted to the left
595         else
596         {
597             //Ensure the helicopter does not cross the boundry of the screen
598             if(heli_page != BOUND_HELI_D)
599                 heli_page++;
600             if(--heli_line == BOUND_HELI_L)
601                 heli_line = 3;
602             GLCD_WriteHeli(HELI,heli_page,heli_line);
603             GLCD_WriteHeli(' ',heli_page - 1, heli_line);
604         }
605     }
606     // Function: int0_isr()
607     // Descrip : This external 0 interrupt service routine allows a user to enter cheat mode
    and allows the user to calibrate the accellerometer
608     void int0_isr(void) __interrupt (0)
609     {
610         //Variables
611         static unsigned char mode = 0;
612 
613         //Ensure calibrate mode is not chosen
614         if(!calibrate_mode)
615         {
616             //Switch between cheat mode and normal mode after every button press
617             if(!mode)
618             {
619                 P1_2 = 0;
620                 cheat_mode = 1;
621                 mode = 1;
622             }
623             else
624             {
625                 P1_2 = 1;
626                 cheat_mode = 0;
627                 mode = 0;
628             }
629         }
630         else
631         {
632             //Set calibrate flag so calibration
633             calibrate = 1;
634             //Clear the calibrate mode flag
```

```
635             calibrate_mode = 0;
636         }
637 
638     }
639 
```

```
// Author     : Ali Ismail
// Description: This files contais initalization sequences
#ifndef INIT_H
#define INIT_H
//Function Definitions
void timer0_init();
void timer1_init();
void PCA_interrupt_init();
void RS232_init();
void PWM_init();
void HS_output_init();
#endif

```

```c
// Author     : Ali Ismail
// Description: This files contais initalization sequences
#include <at89c51ed2.h>  //also includes 8052.h and 8051.h
#include <mcs51reg.h>
#include <8052.h>   // also included in at89c51ed2.h
#include <stdio.h>
#include "init.h"

// Function: timer1_init()
// Descrip : Initializes timer1 for baud rate generator
void timer1_init()
{
    //Timer 1 Init
    TMOD |= 0x20;      //Mode 2 : 8-bit auto-reload timer 1
    TH1 = 0xFA;        //9600 BAUD Rate
    TCON |= 0x40;      //Start timer1
}
// Function: RS232_init()
// Descrip : Initializes serial port
void RS232_init()
{
    //Serial Port Init
    SCON |= 0x50;      //Enable Serial Port,Mode 1 8-bit UART, variable baud rate
    TI = 1;            //Reset the transmit flag
}


```

```
1  // Author     : Ali Ismail
2  // Description: This files contais all the i2c functions that handle i2c transactions
3  // Credits    :
   http://www.8051projects.info/resources/interfacing-i2c-eeprom-with-8051-microcontroller.78/
4  #ifndef I2C_H
5  #define I2C_H
6
7  #include <at89c51ed2.h>
8  #include <mcs51reg.h>
9  #include <8052.h>
10 #include <stdio.h>
11 #include <stdlib.h>
12 #include <mcs51/8051.h>
13
14 //defines
15 #define I2C_DATA_SIZE 8
16 #define SEND_FAILED   1
17 #define SEND_FAILED_CODE 0x100
18 #define SEND_SUCC_CODE   0x200
19 #define CALC_SLAVE_ADDR_WR(x,y) (y | ((x & 0x700) >> 7))
20 #define CALC_SLAVE_ADDR_RD(x,y) (y | ((x & 0x700) >> 7))
21 #define CALC_ADDR(x) (x & 0x00FF)
22 #define EEPROM_ID_W   0xA0
23 #define EEPROM_ID_R   0xA1
24 #define ACCEL_ID_W    0x3A
25 #define ACCEL_ID_R    0x3B
26
27 //Function Definitions
28 __sbit __at (0x94) SCL;
29 __sbit __at (0x95) SDA;
30 void start_i2c();
31 void stop_i2c();
32 void ack();
33 void noack();
34 void i2c_delay();
35 unsigned char send(unsigned char data_in);
36 unsigned char read();
37 int eebytew(int addr, unsigned char databyte);
38 int eebyter(int addr);
39 int accelbytew(int addr, unsigned char databyte);
40 int accelbyter(int addr);
41 int accelbyter_m(int addr, unsigned char *read1, unsigned char *read2);
42 #endif // I2C_H
43
```

```
1  // Author     : Ali Ismail
2  // Description: This files contais all the i2c functions that handle i2c transactions
3  // Credits    : http://www.8051projects.info/resources/interfacing-i2c-eeprom-with-8051-microcontroller.78/
4  #include "i2c.h"
5  #include "terminal_comm.c"
6  #include "lcd.h"
7
8  // Function: start_i2c()
9  // Descrip : Start sequence to begin sending data
10 // Credits : http://www.8051projects.info/resources/interfacing-i2c-eeprom-with-8051-microcontroller.78/
11 //           Everything here is theirs
12 void start_i2c()
13 {
14     //SCL must be high and SDA must transition from high to low
15     //Add delay between transitions
16     SDA = 1;
17     SCL = 1;
18     i2c_delay();
19     SDA = 0;
20     i2c_delay();
21     //SCL = 0;
22 }
23 // Function: stop_i2c()
24 // Descrip : Stop sequence to end i2c transaction
25 // Credits : http://www.8051projects.info/resources/interfacing-i2c-eeprom-with-8051-microcontroller.78/
26 //           Everything here is theirs
27 void stop_i2c()
28 {
29     //SCL must be high and SDA must transition from low to high
30     //Add delay between transitions
31     SDA = 0;
32     SCL = 1;
33     i2c_delay();
34     SDA = 1;
35     i2c_delay();
36 }
37 // Function: ack()
38 // Descrip : Sends an acknowledge bit/compensates for acknowledge bit
39 // Credits : http://www.8051projects.info/resources/interfacing-i2c-eeprom-with-8051-microcontroller.78/
40 //           Everything here is theirs
41 void ack()
42 {
43     //Acknowedge data has been sent by sending a 0
44     SDA = 0;
45     i2c_delay();
46     SCL = 1;
47     i2c_delay();
48     SCL = 0;
49
50 }
51 // Function: noack()
52 // Descrip : Reverse acknoledge
53 // Credits : http://www.8051projects.info/resources/interfacing-i2c-eeprom-with-8051-microcontroller.78/
54 //           Everything here is theirs
55 void noack()
56 {
57     //Acknowedge data has been sent by sending a 0
58     SDA = 1;
59     i2c_delay();
60     SCL = 1;
61     i2c_delay();
62     SCL = 0;
63
64 }
65 // Function: ack_poll()
66 // Descrip : Ensures data write has occured and is ready for the next
67 void ack_poll()
68 {
69     start_i2c();
70     //Poll acknowledge
71     while(send(EEPROM_ID_W) == SEND_FAILED);
72     stop_i2c();
73 }
74 // Function: send()
75 // Input   : data to be sent
76 // Output  : status of send
77 // Descrip : send sequence for i2c
78 // Credits : http://www.8051projects.info/resources/interfacing-i2c-eeprom-with-8051-microcontroller.78/
```

```
79  //           What is my own is returning the status of the send
80  unsigned char send(unsigned char data_in)
81  {
82      int i;
83      unsigned char ack_bit;
84
85      //Set the clock low
86      SCL = 0;
87      for(i = 0; i < I2C_DATA_SIZE; i++)
88      {
89          i2c_delay();
90          //Check the msb of Data and send SDA accordingly
91          SDA = (data_in & 0x80) ? 1:0;
92          //Raise the clock to indicate data is valid
93          SCL = 1;
94          i2c_delay();
95          SCL = 0;
96          //Grab the next bit to be send
97          data_in <<= 1;
98      }
99      //Read acknowledge bit sent by receiver
100     ack_bit = SDA;
101     SCL = 1;
102     i2c_delay();
103     SCL = 0;
104
105     //Acknowledge is a logic low so return the inverse
106     return ack_bit;
107 }
108 // Function: read()
109 // Input   :
110 // Output  : data that was read on the i2c bus
111 // Descrip : read sequence for i2c
112 // Credits :
    http://www.8051projects.info/resources/interfacing-i2c-eeprom-with-8051-microcontroller.78/
113 //           Everything here is theirs
114 unsigned char read()
115 {
116     int i;
117     unsigned char temp = 0;
118     //Set SDA as an input
119     SDA = 1;
120     for(i = 0; i < I2C_DATA_SIZE; i++)
121     {
122         //Transition the clock low to high
123         SCL = 0;
124         i2c_delay();
125         SCL = 1;
126
127         //Shift the value being read to prepare for the next bit to be read
128         temp <<= 1;
129         //If the data read is a 1, then add it to temp, otherwise, place a zero there
130         if(SDA)
131             temp |= 0x01;
132         else
133             temp &= 0xFE;
134     }
135     SCL = 0;
136     return temp;
137 }
138
139 // Function: eebytew()
140 // Input   : Address to write specified data to
141 // Output  : Status of the write
142 // Descrip : write byte to specified address
143 // Credits :
    http://www.8051projects.info/resources/interfacing-i2c-eeprom-with-8051-microcontroller.78/
144 //           slave address and byte address macros and error handling is my own work
145 int eebytew(int addr, unsigned char databyte)
146 {
147     unsigned char slave_address = CALC_SLAVE_ADDR_WR(addr,EEPROM_ID_W);
148     unsigned char byte_addr = CALC_ADDR(addr);
149
150     //Start sequence
151     start_i2c();
152     //Send slave address
153     if(send(slave_address) == SEND_FAILED)
154         return SEND_FAILED_CODE;
155     //Send the address which is to be written to
156     if(send(byte_addr) == SEND_FAILED)
157         return SEND_FAILED_CODE;
158     //Send the data to be written
159     if(send(databyte) == SEND_FAILED)
160         return SEND_FAILED_CODE;
```

```
161         //Stop sequence
162         stop_i2c();
163         //Ensure write has finished
164         ack_poll();
165 
166         return SEND_SUCC_CODE;
167 
168     }
169     // Function: eebyter()
170     // Input    : Address to read from
171     // Output   : Data read
172     // Descrip : Read byte, returns data or status
173     // Credits :
    http://www.8051projects.info/resources/interfacing-i2c-eeprom-with-8051-microcontroller.78/m
174     //           The macros and error handling are mine are mine
175     int eebyter(int addr)
176     {
177         unsigned char slave_address_wr = CALC_SLAVE_ADDR_WR(addr,EEPROM_ID_W);
178         unsigned char slave_address_rd = CALC_SLAVE_ADDR_RD(addr,EEPROM_ID_R);
179         unsigned char byte_addr = CALC_ADDR(addr);
180         unsigned char byte_read;
181 
182         //Random Read, therefore, setup dummy send
183         //Start sequence
184         start_i2c();
185         //Send Slave Address
186         if(send(slave_address_wr) == SEND_FAILED)
187             return SEND_FAILED_CODE;
188         //Send Address that needs to be read later
189         if(send(byte_addr) ==  SEND_FAILED)
190             return SEND_FAILED_CODE;
191         //Dummy sequence finished, start the read
192         start_i2c();
193         //Send Slave Address to do a read
194         if(send(slave_address_rd) == SEND_FAILED)
195             return SEND_FAILED_CODE;
196         //Read from the transmittor
197         byte_read = read();
198         //Let receiver know you are done receiving
199         noack();
200         //Stop sequence
201         stop_i2c();
202 
203         return byte_read;
204     }
205 
206     // Function: accelbytew()
207     // Input    : Address to write specified data to
208     // Output   : Status of the write
209     // Descrip : write byte to specified address
210     int accelbytew(int addr, unsigned char databyte)
211     {
212         unsigned char slave_address = CALC_SLAVE_ADDR_WR(addr,ACCEL_ID_W);
213         unsigned char byte_addr = CALC_ADDR(addr);
214 
215         //Start sequence
216         start_i2c();
217         //Send slave address
218         if(send(slave_address) == SEND_FAILED)
219             return SEND_FAILED_CODE;
220         //Send the address which is to be written to
221         if(send(byte_addr) == SEND_FAILED)
222             return SEND_FAILED_CODE;
223         //Send the data to be written
224         if(send(databyte) == SEND_FAILED)
225             return SEND_FAILED_CODE;
226         //Stop sequence
227         stop_i2c();
228 
229         //Ensure write has finished
230         delay_ms(5);
231         return SEND_SUCC_CODE;
232     }
233     // Function: accelbyter()
234     // Input    : Address to read from
235     // Output   : Data read or status of read
236     // Descrip : Read byte, returns data or status
237     int accelbyter(int addr)
238     {
239         unsigned char slave_address_wr = CALC_SLAVE_ADDR_WR(addr,ACCEL_ID_W);
240         unsigned char slave_address_rd = CALC_SLAVE_ADDR_RD(addr,ACCEL_ID_R);
241         unsigned char byte_addr = CALC_ADDR(addr);
242         unsigned char byte_read;
243 
```

```
244         //Random Read, therefore, setup dummy send
245         //Start sequence
246         start_i2c();
247         //Send Slave Address
248         if(send(slave_address_wr) == SEND_FAILED)
249             return SEND_FAILED_CODE;
250         //Send Address that needs to be read later
251         if(send(byte_addr) ==  SEND_FAILED)
252             return SEND_FAILED_CODE;
253         //Dummy sequence finished, start the read
254         start_i2c();
255         //Send Slave Address to do a read
256         if(send(slave_address_rd) == SEND_FAILED)
257             return SEND_FAILED_CODE;
258         //Read from the transmittor
259         byte_read = read();
260         //Let receiver know you are done receiving
261         noack();
262         //Stop sequence
263         stop_i2c();
264 
265         return byte_read;
266     }
267     // Function: accelbyter_m()
268     // Input   : Address to read from, read 1, and read 2
269     // Descrip : This function does two consecutive reads of the ADXL345
270     int accelbyter_m(int addr, unsigned char *read1, unsigned char *read2)
271     {
272         unsigned char slave_address_wr = CALC_SLAVE_ADDR_WR(addr,ACCEL_ID_W);
273         unsigned char slave_address_rd = CALC_SLAVE_ADDR_RD(addr,ACCEL_ID_R);
274         unsigned char byte_addr = CALC_ADDR(addr);
275 
276         //Random Read, therefore, setup dummy send
277         //Start sequence
278         start_i2c();
279         //Send Slave Address
280         if(send(slave_address_wr) == SEND_FAILED)
281             return SEND_FAILED_CODE;
282         //Send Address that needs to be read later
283         if(send(byte_addr) ==  SEND_FAILED)
284             return SEND_FAILED_CODE;
285         //Dummy sequence finished, start the read
286         start_i2c();
287         //Send Slave Address to do a read
288         if(send(slave_address_rd) == SEND_FAILED)
289             return SEND_FAILED_CODE;
290         //Read from the transmittor
291         *read1 = read();
292         //Send acknowledge
293         ack();
294         //Read from the transmittor
295         *read2 = read();
296         //Let receiver know you are done receiving
297         noack();
298         //Stop sequence
299         stop_i2c();
300 
301         return 0;
302     }
303     // Function: i2c_delay()
304     // Descrip : Short delay for toggling SCL
305     void i2c_delay()
306     {
307             __asm
308                 nop
309                 nop
310                 nop
311             __endasm;
312     }
313 
314 
```

```
// Author     : Ali Ismail
// Description: This files contais all the lcd functions that handle lcd transactions and screen drawings
// Credits    : http://jormungand.net/projects/devices/avrlcd/ and http://en.radzio.dxp.pl/ks0108/

#ifndef LCD_H
#define LCD_H

#include <at89c51ed2.h>
#include <mcs51reg.h>
#include <8052.h>
#include <stdio.h>
#include <stdlib.h>
#include <mcs51/8051.h>

//Write and read commands for GLCD
extern __xdata __at(0xF000) unsigned char LCD_WR;
extern __xdata __at(0xF100) unsigned char LCD_RD;
//Globals
extern unsigned char calibrate_mode;
extern unsigned char calibrate;
//Defines
/*****************CREDIT START*********************/
// Credits : http://jormungand.net/projects/devices/avrlcd/
#define LCD_SELECT_CS1     (CS1 = 0, CS2 = 1)
#define LCD_SELECT_CS2     (CS2 = 0, CS1 = 1)
#define LCD_DATA           1
#define LCD_INST           0
#define LCD_SELECT_DATA    (RS = 1)
#define LCD_SELECT_INST    (RS = 0)
#define LCD_SELECT_CHIP(A)((A) ? LCD_SELECT_CS2 : LCD_SELECT_CS1);
#define LCD_SELECT_REG(A) ((A) ? LCD_SELECT_DATA : LCD_SELECT_INST);
#define LCD_BUSY           0x80
#define LCD_RESET          0x10
#define LCD_POWERON(P)     (((P)?1:0) | 0x3E)
#define LCD_STARTLINE(L)   (((L) & 0x3F) | 0xC0)
#define LCD_YADDR(Y)       (((Y) & 0x3F) | 0x40)
#define LCD_XADDR(X)       (((X) & 0x07) | 0xB8)
/*****************CREDIT END***********************/
#define BRICK              485
#define SOLID_BRICK        492
#define CLEAR_LINE         1
#define INVERT             1
#define NORMAL             0
#define ASCII_OFFSET       0x30
#define HELI               128
#define TOP_OPT            2
#define BOTTOM_OPT         5
#define MOTOR_ON           P1_7 = 1
#define MOTOR_OFF          P1_7 = 0

//GLCD Control signals
__sbit __at (0x93) RS;
__sbit __at (0x91) CS1;
__sbit __at (0x90) CS2;
__sbit __at (0x96) RST;

//Function Declarations
void lcdinit();
unsigned char lcd_read(unsigned chip, unsigned reg);
void lcd_write(unsigned char chip, unsigned char reg, unsigned char data_in);
void lcd_wait(unsigned char chip);
void lcd_write_wait(unsigned char chip, unsigned char reg, unsigned char data_in);
void lcd_clear();
void lcd_clear_invert();
void clear_game_screen();
void still_screen();
void accel_screen();
void creater_screen();
void cheater();
void game_over();
unsigned char game_menu();
unsigned char difficulty_menu();
void draw_banner();
void next_level_screen(unsigned char level);
void GLCD_WriteChar(char charToWrite, unsigned char page, unsigned char line, unsigned char invert);
void GLCD_WriteHeli(char charToWrite, unsigned char page, unsigned char line);
void GLCD_WriteBrick(unsigned int charToWrite, unsigned char page, int line, unsigned char solid);
void GLCD_WriteString(char * stringToWrite, unsigned char page, unsigned char line, unsigned char invert);
void draw_pillar(unsigned char len, unsigned char bott_up, int line, unsigned char solid);
```

```
80  void delay_ms(int num_ms);
81  unsigned int rand(int min_num, int max_num);
82  #endif // LCD_H
83
```

```
1  // Author     : Ali Ismail
2  // Description: This files contais all the lcd functions that handle lcd transactions and
   screen drawings
3  // Credits    : http://jormungand.net/projects/devices/avrlcd/ and
   http://en.radzio.dxp.pl/ks0108/
4
5  #include <at89c51ed2.h>  //also includes 8052.h and 8051.h
6  #include <mcs51reg.h>
7  #include <8052.h>   // also included in at89c51ed2.h
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <mcs51/8051.h>
11 #include "terminal_comm.h"
12 #include "lcd.h"
13 #include "font5x8.h"
14 #include "ADXL345.h"
15
16 //
17 __xdata __at(0xF000) unsigned char LCD_WR;
18 __xdata __at(0xF100) unsigned char LCD_RD;
19
20 // Function: lcdinit()
21 // Descrip : Initializes the LCD
22 // Credits : http://jormungand.net/projects/devices/avrlcd/
23 //           This init routine is entirely his
24 void lcdinit()
25 {
26     //Turn off both sections of GLCD
27     RS = 0;
28     CS1 = 1;
29     CS2 = 1;
30
31     delay_ms(10);
32
33     lcd_wait(0);
34     lcd_wait(1);
35     //Turn on both sections of GLCD
36     lcd_write_wait(0, LCD_INST, LCD_POWERON(1));
37     lcd_write_wait(1, LCD_INST, LCD_POWERON(1));
38     lcd_write_wait(0, LCD_INST, LCD_STARTLINE(0));
39     lcd_write_wait(1, LCD_INST, LCD_STARTLINE(0));
40     //Clear the entire GLCD screen
41     lcd_clear();
42 }
43 // Function: lcd_read()
44 // Input   : section of the LCD and register (data or instruction)
45 // Output  : data/status from/of the lcd
46 // Descrip : Reads data or status of the LCD
47 // Credits : http://jormungand.net/projects/devices/avrlcd/
48 //           I removed all enables and delays between enable since I
49 //           implemented reads and writes to the LCD differently. I
50 //           am using A8 (high byte address) to set/clear R/W
51 unsigned char lcd_read(unsigned chip, unsigned reg)
52 {
53     unsigned char val;
54
55     //Select the CS
56     //Select the data or instruction register
57     LCD_SELECT_CHIP(chip)
58     LCD_SELECT_REG(reg)
59     val = LCD_RD;
60
61     return val;
62 }
63 // Function: lcd_write()
64 // Input   : section of the LCD, register (data or instruction), and data to be written
65 // Descrip : Writes data/commands to the LCD
66 // Credits : http://jormungand.net/projects/devices/avrlcd/
67 //           I removed all enables and delays between enable since I
68 //           implemented reads and writes to the LCD differently. I
69 //           am using A8 (high byte address) to set/clear R/W
70 void lcd_write(unsigned char chip, unsigned char reg, unsigned char data_in)
71 {
72     //Select the CS
73     //Select the data or instruction register
74     LCD_SELECT_CHIP(chip);
75     LCD_SELECT_REG(reg);
76     LCD_WR = data_in;
77 }
78 // Function: lcd_wait()
79 // Input   : section of the LCD to wait for
80 // Descrip : Waits until the LCD is ready for data/command
81 // Credits : http://jormungand.net/projects/devices/avrlcd/
82 //           This is entirely his
```

```
83  void lcd_wait(unsigned char chip)
84  {
85      //Wait till the LCD is not busy or in reset
86      while(lcd_read(chip, LCD_INST) & (LCD_BUSY | LCD_RESET)) { };
87  }
88  // Function: lcd_write_wait()
89  // Input   : section of the LCD, register (data or instruction), and data to be written
90  // Descrip : Waits until the LCD is ready for data/command and then does a write to the LCD
91  // Credits : http://jormungand.net/projects/devices/avrlcd/
92  //           This is entirely his
93  void lcd_write_wait(unsigned char chip, unsigned char reg, unsigned char data_in)
94  {
95      lcd_wait(chip);
96      lcd_write(chip, reg, data_in);
97  }
98  // Function: lcd_clear()
99  // Descrip : Writes a 0 to to all the pixels on screen
100 // Credits : http://jormungand.net/projects/devices/avrlcd/
101 //           This is entirely his
102 void lcd_clear()
103 {
104     //Variables
105     unsigned char x, y;
106
107     //For CS1
108     //For each page
109     for(x = 0; x < 8; ++x)
110     {
111         //Set page and set the y address to the beginning
112         lcd_write_wait(0, LCD_INST, LCD_YADDR(0));
113         lcd_write_wait(0, LCD_INST, LCD_XADDR(x));
114         //For each column of 8 pixels
115         for(y = 0; y < 64; ++y)
116         {
117             //Set 8 pixels to 0s
118             lcd_write_wait(0, LCD_DATA, 0);
119         }
120     }
121     //For CS2
122     //For each page
123     for(x = 0; x < 8; ++x)
124     {
125         //Set page and set the y address to the beginning
126         lcd_write_wait(1, LCD_INST, LCD_YADDR(0));
127         lcd_write_wait(1, LCD_INST, LCD_XADDR(x));
128         //For each column of 8 pixels
129         for(y = 0; y < 64; ++y)
130         {
131             //Set 8 pixels to 0s
132             lcd_write_wait(1, LCD_DATA, 0);
133         }
134     }
135 }
136 // Function: lcd_clear_invert()
137 // Descrip : Writes a 1 to to all the pixels on screen
138 // Credits : http://jormungand.net/projects/devices/avrlcd/
139 //           Same as lcd_clear, but I write 1's this time
140 void lcd_clear_invert()
141 {
142     //Variables
143     unsigned char x, y;
144
145     //For CS1
146     //For each page
147     for(x = 0; x < 8; ++x)
148     {
149         //Set page and set the y address to the beginning
150         lcd_write_wait(0, LCD_INST, LCD_YADDR(0));
151         lcd_write_wait(0, LCD_INST, LCD_XADDR(x));
152         //For each column of 8 pixels
153         for(y = 0; y < 64; ++y)
154         {
155             //Set 8 pixels to 1s
156             lcd_write_wait(0, LCD_DATA, 0xff);
157         }
158     }
159     //For CS2
160     //For each page
161     for(x = 0; x < 8; ++x)
162     {
163         //Set page and set the y address to the beginning
164         lcd_write_wait(1, LCD_INST, LCD_YADDR(0));
165         lcd_write_wait(1, LCD_INST, LCD_XADDR(x));
166         //For each column of 8 pixels
```

```
167             for(y = 0; y < 64; ++y)
168             {
169                 //Set 8 pixels to 1s
170                 lcd_write_wait(1, LCD_DATA, 0xff);
171             }
172         }
173     }
174     // Function: clear_game_screen()
175     // Descrip : Clears screen in between pages 1 and 6
176     // Credits : http://jormungand.net/projects/devices/avrlcd/
177     //           Same as lcd_clear, but I only clear from pages 1 to 6
178     void clear_game_screen()
179     {
180         //Variables
181         unsigned char x, y;
182         //For CS1
183         //For each page
184         for(x = 1; x < 7; ++x)
185         {
186             //Set page and set the y address to the beginning
187             lcd_write_wait(0, LCD_INST, LCD_YADDR(0));
188             lcd_write_wait(0, LCD_INST, LCD_XADDR(x));
189             //For each column of 8 pixels
190             for(y = 0; y < 64; ++y)
191             {
192                 //Set 8 pixels to 0s
193                 lcd_write_wait(0, LCD_DATA, 0x00);
194             }
195         }
196         //For CS2
197         //For each page
198         for(x = 1; x < 7; ++x)
199         {
200             //Set page and set the y address to the beginning
201             lcd_write_wait(1, LCD_INST, LCD_YADDR(0));
202             lcd_write_wait(1, LCD_INST, LCD_XADDR(x));
203             //For each column of 8 pixels
204             for(y = 0; y < 64; ++y)
205             {
206                 //Set 8 pixels to 0s
207                 lcd_write_wait(1, LCD_DATA, 0x00);
208             }
209         }
210     }
211     // Function: still_screen()
212     // Descrip : Displays the game screen that never changes
213     void still_screen()
214     {
215         //Game strings
216         unsigned char score[10] = {"SCORE: "};
217         unsigned char title[10] = {"HELI_TILT"};
218 
219         //Draw black banner on top and bottom of screen
220         //Each is 8 x 128
221         draw_banner();
222 
223         //Draw the strings onto the game screen
224         GLCD_WriteString(score,0,3, INVERT);
225         GLCD_WriteString(title,7,40, INVERT);
226     }
227     // Function: next_level_screen()
228     // Input   : Level that has been completed
229     // Descrip : This function handles letting the game know which level they
230     //           completed and which level they are starting
231     void next_level_screen(unsigned char level)
232     {
233         //Variables
234         unsigned char level_complete[] = {"COMPLETED LEVEL "};
235         unsigned char level_begin[] = {"START LEVEL "};
236         unsigned char level_end[] = {"FINAL LEVEL >:) "};
237         unsigned char clear_line[10] = {"         "};
238         unsigned char level_on;
239         unsigned char level_start;
240 
241         //Convert current level to ascii char
242         level_on = level + ASCII_OFFSET;
243         clear_game_screen();
244 
245         //If the gamer is on the final level
246         if(level == 9)
247         {
248             //Tell the gamer they completed level 9
249             GLCD_WriteString(level_complete, 3, 12, NORMAL);
250             GLCD_WriteChar(level_on, 3, 107, NORMAL);
```

```
            //Delay to allow gamer time to read message
            delay_ms(6000);
            clear_game_screen();
            //Tell game they are starting the final level
            GLCD_WriteString(level_end, 3, 20, NORMAL);
            //Delay to allow gamer time to read message
            delay_ms(6000);
        }
        //If gamer is on first level
        else if (level == 0)
        {
            //Tell the gamer they are starting level 1
            GLCD_WriteString(level_begin, 3, 25, NORMAL);
            level_on++;
            GLCD_WriteChar(level_on, 3, 95, NORMAL);
            //Delay to allow gamer time to read message
            delay_ms(6000);
        }
        //If the gamer is in between level 2 and 9
        else
        {
            //Convert start level to ascii char
            level_start = level + 1 + ASCII_OFFSET;
            //Tell the gamer the level they completed
            GLCD_WriteString(level_complete, 3, 12, NORMAL);
            GLCD_WriteChar(level_on, 3, 107, NORMAL);
            //Delay to allow gamer time to read message
            delay_ms(6000);
            clear_game_screen();
            //Tell the gamer the level they are starting
            GLCD_WriteString(level_begin, 3, 25, NORMAL);
            GLCD_WriteChar(level_start, 3, 95, NORMAL);
            //Delay to allow gamer time to read message
            delay_ms(6000);
        }
        clear_game_screen();

    }
    // Function: game_over()
    // Descrip : Displays game over
    void game_over()
    {
        //Variables
        unsigned char game_finished[] = {"GAME OVER :("};
        clear_game_screen();
        //Display that the game is over
        GLCD_WriteString(game_finished, 3, 30, NORMAL);
        //Delay to allow gamer time to read message
        delay_ms(6000);
        clear_game_screen();
    }
    // Function: cheater()
    // Descrip : Displays that the gamer is a cheater
    void cheater()
    {
        unsigned char game_finished[] = {"CHEATER! >:)"};
        clear_game_screen();
        //Display cheater
        GLCD_WriteString(game_finished, 3, 30, NORMAL);
        //Delay to allow gamer time to read message
        delay_ms(6000);
        clear_game_screen();
    }
    // Function: creater_screen()
    // Descrip : Displays the credits of the game
    void creater_screen()
    {
        //Game strings
        unsigned char creater[] = {"ALI KILLUMINATI"};
        unsigned char presents[] = {"PRESENTS"};
        unsigned char smurf_cat[] = {"A SMURFCAT"};
        unsigned char production[] = {"PRODUCTION"};
        unsigned char heli_tilt[] = {"HELI_TILT"};
        int x;

        clear_game_screen();

        //Draw black banner on top and bottom of screen
        //Each is 8 x 128
        draw_banner();

         //Display game title in the banner
        GLCD_WriteString(heli_tilt,0,40,INVERT);
        GLCD_WriteString(heli_tilt,7,40,INVERT);
```

```
335         //Diplay creater
336         GLCD_WriteString(creater, 3, 15, NORMAL);
337         GLCD_WriteString(presents, 4, 40, NORMAL);
338         //Delay to allow gamer time to read message
339         delay_ms(5000);
340         clear_game_screen();
341         //Display production
342         GLCD_WriteString(smurf_cat, 3, 35, NORMAL);
343         GLCD_WriteString(production, 4, 35,NORMAL);
344         //Delay to allow gamer time to read message
345         delay_ms(5000);
346         clear_game_screen();
347         //Display game name
348         GLCD_WriteString(heli_tilt, 3, 40, NORMAL);
349         GLCD_WriteHeli(HELI, 3, 64);
350         //Delay to allow gamer time to read message
351         delay_ms(5000);
352 
353         MOTOR_ON;
354         //Have helicopter glide to the end of the screen
355         for(x = 0; x < 65; x++)
356         {
357             GLCD_WriteHeli(HELI, 4, 59 + x);
358             delay_ms(100);
359         }
360         MOTOR_OFF;
361         clear_game_screen();
362 
363     }
364     // Function: game_menu()
365     // Output  : returns menu choice of gamer
366     // Descrip : Displays the game menu options play game, high scores, calibrate and set
    difficulty
367     //           and captures which option the user would like
368     unsigned char game_menu()
369     {
370         //Game strings
371         unsigned char play_game[] = {"PLAY GAME"};
372         unsigned char high_scores[] = {"HIGH SCORES"};
373         unsigned char calibrate[] = {"CALIBRATE"};
374         unsigned char heli_tilt[] = {"HELI_TILT"};
375         unsigned char diffculty[] = {"SET DIFFICULTY"};
376         int x = 0, y = 0, choice_page = 2, y_count = 0;
377 
378         clear_game_screen();
379         //Draw black banner on top and bottom of screen
380         //Each is 8 x 128
381         draw_banner();
382 
383         //Display game title in banner
384         GLCD_WriteString(heli_tilt,0,40,INVERT);
385         GLCD_WriteString(heli_tilt,7,40,INVERT);
386         //Draw game options
387         //Draw helicopter next to option, which will be used by user
388         //to naviage game menu
389         GLCD_WriteHeli(HELI,choice_page,15);
390         GLCD_WriteString(play_game,2,40,NORMAL);
391         GLCD_WriteString(diffculty,3,25,NORMAL);
392         GLCD_WriteString(high_scores,4,35,NORMAL);
393         GLCD_WriteString(calibrate,5,40,NORMAL);
394 
395         //Slight delay to account from page transition
396         delay_ms(4000);
397 
398         while(1)
399         {
400             //Read orientation of conroller
401             y = ADXL345_read_y();
402             x = ADXL345_read_x();
403             //If gamer tilts up
404             if(y <= RIGHT_UP_MIN)
405             {
406                 //Erase the current position of the helicopter
407                 GLCD_WriteChar(' ', choice_page, 15,NORMAL);
408                 //Ensure helicopter does not go past first option
409                 if(--choice_page < TOP_OPT)
410                     choice_page = TOP_OPT;
411             }
412             //If gamer tilts down
413             else if( y >= LEFT_DOWN_MIN)
414             {
415                 //Erase the current position of the helicopter
416                 GLCD_WriteChar(' ', choice_page, 15,NORMAL);
417                 //Ensure helicopter does not go past last option
```

```
418             if(++choice_page > BOTTOM_OPT)
419                 choice_page = BOTTOM_OPT;
420         }
421         //Draw the helicopter based on controller orientation
422         GLCD_WriteHeli(HELI,choice_page,15);
423         //Delay to allow user to process their menu navigation
424         delay_ms(750);
425         //If the user tilts right, they choose this menu option
426         if(x <= RIGHT_UP_MIN)
427             return choice_page - 1;
428 
429     }
430 }
431 // Function: difficulty_menu()
432 // Output  : returns difficulty choice of the user
433 // Descrip : Displays the game difficulty options and allows user to navigate and choose
434 unsigned char difficulty_menu()
435 {
436     //Game strings
437     unsigned char easy[] = {"TOO EASY"};
438     unsigned char medium[] = {"TOO MEDIUM"};
439     unsigned char smash_cat[] = {"SMASHING CAT"};
440     unsigned char heli_tilt[] = {"HELI_TILT"};
441     int x = 0, y = 0, choice_page = 2, y_count = 0;
442 
443     clear_game_screen();
444     //Draw black banner on top and bottom of screen
445     //Each is 8 x 128
446     draw_banner();
447 
448     //Display game title
449     GLCD_WriteString(heli_tilt,0,40,INVERT);
450     GLCD_WriteString(heli_tilt,7,40,INVERT);
451     //Draw difficulty options
452     //Draw helicopter next to option, which will be used by user
453     //to naviage difficulties
454     GLCD_WriteHeli(HELI,choice_page,15);
455     GLCD_WriteString(easy,2,40,NORMAL);
456     GLCD_WriteString(medium,3,35,NORMAL);
457     GLCD_WriteString(smash_cat,4,30,NORMAL);
458     GLCD_WriteHeli(HELI,4,78);
459 
460     //Slight delay to account from page transition
461     delay_ms(4000);
462 
463     while(1)
464     {
465         //Read orientation of conroller
466         y = ADXL345_read_y();
467         x = ADXL345_read_x();
468         //If gamer tilts up
469         if(y <= RIGHT_UP_MIN)
470         {
471             //Erase the current position of the helicopter
472             GLCD_WriteChar(' ', choice_page, 15,NORMAL);
473             //Ensure helicopter does not go past first option
474             if(--choice_page < TOP_OPT)
475                 choice_page = TOP_OPT;
476         }
477         else if( y >= LEFT_DOWN_MIN)
478         {
479             //Erase the current position of the helicopter
480             GLCD_WriteChar(' ', choice_page, 15,NORMAL);
481             //Ensure helicopter does not go past last option
482             if(++choice_page > BOTTOM_OPT - 1)
483                 choice_page = BOTTOM_OPT - 1;
484         }
485         //Draw the helicopter based on controller orientation
486         GLCD_WriteHeli(HELI,choice_page,15);
487         //Delay to allow user to process their menu navigation
488         delay_ms(750);
489         //If the user tilts right, they choose this menu option
490         if(x <= RIGHT_UP_MIN)
491             return choice_page - 2;
492 
493     }
494 }
495 // Function: draw_banner()
496 // Descrip : Displays a banner on the top and bottom of the GLCD, each are
497 //           a page tall and they go from end to end wide.
498 void draw_banner()
499 {
500     //Variables
501     int x, y;
```

```
502       //Left screen page 0
503       for(x = 0; x < 1; x++)
504       {
505           //Set page and set the y address to the beginning
506           lcd_write_wait(0, LCD_INST, LCD_YADDR(0));
507           lcd_write_wait(0, LCD_INST, LCD_XADDR(x));
508           //Draw 8 pixels of 1's
509           for(y = 0; y < 64; ++y)
510             lcd_write_wait(0, LCD_DATA, 0xFF);
511       }
512
513       //Left screen page 7
514       for(x = 7; x < 8; x++)
515       {
516           //Set page and set the y address to the beginning
517           lcd_write_wait(0, LCD_INST, LCD_YADDR(0));
518           lcd_write_wait(0, LCD_INST, LCD_XADDR(x));
519           //Draw 8 pixels of 1's
520           for(y = 0; y < 64; ++y)
521             lcd_write_wait(0, LCD_DATA, 0xFF);
522       }
523
524       //Right screen page 0 - 1
525       for(x = 0; x < 1; x++)
526       {
527           //Set page and set the y address to the beginning
528           lcd_write_wait(1, LCD_INST, LCD_YADDR(0));
529           lcd_write_wait(1, LCD_INST, LCD_XADDR(x));
530           //Draw 8 pixels of 1's
531           for(y = 0; y < 64; ++y)
532             lcd_write_wait(1, LCD_DATA, 0xFF);
533       }
534
535       //Right screen page 6 - 7
536       for(x = 7; x < 8; x++)
537       {
538           //Set page and set the y address to the beginning
539           lcd_write_wait(1, LCD_INST, LCD_YADDR(0));
540           lcd_write_wait(1, LCD_INST, LCD_XADDR(x));
541           //Draw 8 pixels of 1's
542           for(y = 0; y < 64; ++y)
543             lcd_write_wait(1, LCD_DATA, 0xFF);
544       }
545   }
546   // Function: accel_screen()
547   // Descrip : Displays measurements of x,y, and z axis of accelerometer in real time.
548   //           It also allows a gamer to calibrate the accelerometer via pushbutton
549   void accel_screen()
550   {
551       //Game strings
552       unsigned char title[20] = {"ADXL345 Readings"};
553       unsigned char x_axis[10] = {"X-Axis:"};
554       unsigned char y_axis[10] = {"Y-Axis:"};
555       unsigned char z_axis[10] = {"Z-Axis:"};
556       unsigned char clear_line[10] = {"        "};
557       unsigned char print_x[10];
558       unsigned char print_y[10];
559       unsigned char print_z[10];
560
561       int x = 0;
562       int y;
563       int z;
564
565       //Invert the display screen
566       lcd_clear_invert();
567       //Draw labels for each axis
568       GLCD_WriteString(title,0,17, INVERT);
569       GLCD_WriteString(x_axis,2,3, INVERT);
570       GLCD_WriteString(y_axis,4,3, INVERT);
571       GLCD_WriteString(z_axis,6,3, INVERT);
572       //Delay to allow gamer time to read message
573       delay_ms(2000);
574
575       //Display the readings of the accelerometer until the user quits or calibrates
576       while(!calibrate && x > RIGHT_UP_MIN)
577       {
578           //Read orientation of conroller
579           x = ADXL345_read_x();
580           y = ADXL345_read_y();
581           z = ADXL345_read_z();
582
583           //Convert each reading to ascii then print to the GLCD
584           _itoa(x,print_x,10);
585           GLCD_WriteString(print_x,2,50, INVERT);
```

```
586             _itoa(y,print_y,10);
587             GLCD_WriteString(print_y,4,50, INVERT);
588             _itoa(z,print_z,10);
589             GLCD_WriteString(print_z,6,50, INVERT);
590             //Add delay so the readings look readable on the GLCD
591             delay_ms(350);
592             //Clear the readings written to GLCD
593             GLCD_WriteString(clear_line,2,50,INVERT);
594             GLCD_WriteString(clear_line,4,50,INVERT);
595             GLCD_WriteString(clear_line,6,50,INVERT);
596         }
597         //Calibrate flag is set when the user presses the push button
598         if(calibrate)
599             ADXL345_calibrate();
600         //Reset calibration flag.
601         calibrate = 0;
602 
603     }
604     // Function: GLCD_WriteChar()
605     // Input    : a char to write, page, starting column, and flag to indicate to invert output
606     // Descrip : Writes a char to the GLCD at a specified page and column. Also, can be
        inverted.
607     // Credits : http://en.radzio.dxp.pl/ks0108/
608     //           Basic idea comes from source, however, I introduced the concept of specifying
        the page
609     //           and column. I also added the ability to invert the character.
610     void GLCD_WriteChar(char charToWrite, unsigned char page, unsigned char line, unsigned char
         invert)
611     {
612         //Determine which chip and column to write to
613         unsigned char lcd_chip = (line & 0x40) ? 1 : 0;
614         unsigned char lcd_y = (line & 0x3F);
615 
616         int i;
617         //Determine the index into font5x8
618         charToWrite -= 32;
619 
620         //Check if the character should be inverted or not
621         if(invert)
622         {
623             //For each column in the character
624             for(i = 0; i < 5; i++)
625             {
626                 //Set the page and column
627                 //Then write the inverse data
628                 lcd_write_wait(lcd_chip, LCD_INST, LCD_YADDR(lcd_y));
629                 lcd_write_wait(lcd_chip, LCD_INST, LCD_XADDR(page));
630                 lcd_write_wait(lcd_chip, LCD_DATA, ~font5x8[(charToWrite * 5 + i)]);
631                 //Determine which chip and column to write to for the next line
632                 lcd_chip = (++line & 0x40) ? 1 : 0;
633                 lcd_y = (line & 0x3F);
634             }
635         }
636         else
637         {
638             for(i = 0; i < 5; i++)
639             {
640                 //Set the page and column
641                 //Then write the data
642                 lcd_write_wait(lcd_chip, LCD_INST, LCD_YADDR(lcd_y));
643                 lcd_write_wait(lcd_chip, LCD_INST, LCD_XADDR(page));
644                 lcd_write_wait(lcd_chip, LCD_DATA, font5x8[(charToWrite * 5 + i)]);
645                 //Determine which chip and column to write to for the next line
646                 lcd_chip = (++line & 0x40) ? 1 : 0;
647                 lcd_y = (line & 0x3F);
648             }
649         }
650     }
651     // Function: GLCD_WriteBrick()
652     // Input    : a char to write, page, starting column, and flag to indicate if the brick
        will have a clear buffer
653     // Descrip : This function draws a brick for a pillar at a specific page and column
654     void GLCD_WriteBrick(unsigned int charToWrite, unsigned char page, int line, unsigned char
        solid)
655     {
656         //Determine which chip and column to write to
657         unsigned char lcd_chip = (line & 0x40) ? 1 : 0;
658         unsigned char lcd_y = (line & 0x3F);
659         unsigned char char_lines;
660         int i;
661 
662         //A solid brick is only 5 lines
663         //a self erasing block is 7 lines
664         if(!solid)
```

```
665             char_lines = 7;
666         else
667             char_lines = 5;
668 
669         //For each line
670         for(i = 0; i < char_lines; i++)
671         {
672             //Check if the brick line is not in the boundry of the screen
673             //Quit loop if it is not in boundry
674             if(line > 127)
675                 break;
676             //Check if the brick line is in boundry
677             if(line >= 0)
678             {
679                 //Set the page and column, then write a line of the brick to the GLCD
680                 lcd_write_wait(lcd_chip, LCD_INST, LCD_YADDR(lcd_y));
681                 lcd_write_wait(lcd_chip, LCD_INST, LCD_XADDR(page));
682                 lcd_write_wait(lcd_chip, LCD_DATA, ~font5x8[(charToWrite + i)]);
683             }
684             //Determine which chip and column to write to for the next line
685             lcd_chip = (++line & 0x40) ? 1 : 0;
686             lcd_y = (line & 0x3F);
687 
688         }
689     }
690     // Function: GLCD_WriteHeli()
691     // Input    : a char to write, page, and starting column
692     // Descrip : This function a helicopter at a specified page and column
693     void GLCD_WriteHeli(char charToWrite, unsigned char page, unsigned char line)
694     {
695         //Determine which chip and column to write to
696         unsigned char lcd_chip = (line & 0x40) ? 1 : 0;
697         unsigned char lcd_y = (line & 0x3F);
698 
699         int i;
700         charToWrite -= 32;
701 
702         for(i = 0; i < 5; i++)
703         {
704             //Set the page and column, then write a line of the helicopter to the GLCD
705             lcd_write_wait(lcd_chip, LCD_INST, LCD_YADDR(lcd_y));
706             lcd_write_wait(lcd_chip, LCD_INST, LCD_XADDR(page));
707             lcd_write_wait(lcd_chip, LCD_DATA, font5x8[(charToWrite * 5 + i)]);
708             //Determine which chip and column to write to for the next line
709             lcd_chip = (++line & 0x40) ? 1 : 0;
710             lcd_y = (line & 0x3F);
711         }
712     }
713     // Function: GLCD_WriteString()
714     // Input    : a char to write, page, and starting column
715     // Descrip : This function writes a string starting at a specified page and column
716     // Credits : http://en.radzio.dxp.pl/ks0108/
717     //           Basic function is theirs, but I added constraints to ensure string is spaced
    out correctly
718     //           and wraps around to the next line
719     void GLCD_WriteString(char * stringToWrite, unsigned char page, unsigned char line,
    unsigned char invert)
720     {
721         int i = 0;
722         while(*stringToWrite)
723         {
724             GLCD_WriteChar(*stringToWrite++,page,line, invert);
725             //Space out the next character 6 spaces over
726             line += 6;
727             //Ensure only 21 character are written to a single page
728             if(++i == 21)
729             {
730                 //Wrap character to the top if there is no more room on the GLCD, otherwise,
731                 //go to the next line
732                 if(++page == 8)
733                     page = 0;
734                 //Reset start column and character count for the page
735                 line = 3;
736                 i = 0;
737             }
738         }
739     }
740     // Function: draw_pillar()
741     // Input    : pillar length, pillar orientation, column, and if it is solid or has a
    clearing boundry
742     // Descrip : This draws pillars that on the floor or ceiling of the GLCD. Also, the
    function draws the pillar
743     //           at the specified size, column, and solid or clearing
744     void draw_pillar(unsigned char len, unsigned char bott_up, int line, unsigned char solid)
```

```
745   {
746       //Variables
747       int i;
748   
749       //Check if the pillar is solid or clearing
750       if(!solid)
751       {
752           //Check if the pillar is on the ceiling or floor
753           if(bott_up)
754           {
755               //Draw pillar
756               for(i = 1; i < len + 1; i++)
757                   GLCD_WriteBrick(BRICK, i,line, solid);
758           }
759           else
760           {
761               //Draw pillar
762               for(i = 6; i > 6 - len; i--)
763                   GLCD_WriteBrick(BRICK, i,line, solid);
764           }
765       }
766       else
767       {
768           //Check if the pillar is on the ceiling or floor
769           if(bott_up)
770           {
771               //Draw pillar
772               for(i = 1; i < len + 1; i++)
773                   GLCD_WriteBrick(SOLID_BRICK, i,line + CLEAR_LINE, solid);
774           }
775           else
776           {
777               //Draw pillar
778               for(i = 6; i > 6 - len; i--)
779                   GLCD_WriteBrick(SOLID_BRICK, i,line + CLEAR_LINE, solid);
780           }
781       }
782   }
783   //Function: delay_ms()
784   //Inputs  : number of ms for delay
785   //Descrip : This functions delays for a given amount of ms
786   void delay_ms(int num_ms)
787   {
788       unsigned int i;
789       unsigned int j;
790       unsigned int k = 0;
791   
792       //do millisecond routine for num_ms milliseconds
793       for(i = 0; i < num_ms; i++)
794           {
795               //a millisecond long
796               for(j = 0; j < 83; j++)
797               {
798                   // do bogus work to waste time
799                   __asm nop __endasm;
800                   k++;
801   
802               }
803           }
804   }
805   //Function: rand()
806   //Inputs  : minimum and maximum number
807   //Descrip : Generates a random number between two numbers
808   //Credts  :
      http://stackoverflow.com/questions/7602919/how-do-i-generate-random-numbers-without-rand-fun
      ction
809   //        All is theirs
810   unsigned int rand(int min_num, int max_num)
811   {
812       unsigned int bits;
813       static unsigned int LFSR = 0xACE1;
814       bits  = ((LFSR >> 0) ^ (LFSR >> 2) ^ (LFSR >> 3) ^ (LFSR >> 5) ) & 1;
815       LFSR =  (LFSR >> 1) | (bits << 15);
816   
817       return (LFSR % (max_num - min_num)) + min_num;
818   }
819
```

```
// Author     : Ali Ismail
// Description: This files contais all the ADXL345 drivers and functions that handle
ADXL345 transactions
#ifndef ADXL345_H
#define ADXL345_H
//defines
#define DEVID           0x00
#define POWER_CTL       0x2D
#define FIFO_CTL        0x38
#define INT_ENABLE      0x2E
#define DATA_FORMAT     0x31
#define BW_RATE         0x2C
#define MEASURE_E       3
#define BYPASS_E        0x1F
#define ENABLE          1
#define DISABLE         0
#define X_OFFSET        0x1E
#define Y_OFFSET        0x1F
#define Z_OFFSET        0x20
#define DATAX_0         0x32
#define DATAX_1         0x33
#define DATAY_0         0x34
#define DATAY_1         0x35
#define DATAZ_0         0x36
#define DATAZ_1         0x37
#define STILL_MIN       -149
#define STILL_MAX       149
#define RIGHT_UP_MIN    -150
#define LEFT_DOWN_MIN   150

//Function Definitions
void ADXL345_init();
void ADXL345_set_bit(int addr, unsigned char bit_ofst, unsigned char bit_val);
unsigned char ADXL345_error_handle_read(int addr);
void ADXL345_error_handle_read_m(int addr, unsigned char *read1, unsigned char *read2);
void ADXL345_error_handle_write(int addr, unsigned char databyte);
void ADXL345_getoffset(unsigned char *x, unsigned char *y, unsigned char *z);
void ADXL345_enable_bypass();
void ADXL345_disable_ints();
void format_data(unsigned char format);
void ADXL345_set_rate(unsigned char rate);
int ADXL345_read_x();
int ADXL345_read_y();
int ADXL345_read_z();
void ADXL345_calibrate();
#endif // ADXL345_H

```

```c
// Author     : Ali Ismail
// Description: This files contais all the ADXL345 drivers and functions that handle ADXL345 transactions
#include <at89c51ed2.h>
#include <mcs51reg.h>
#include <8052.h>
#include <stdio.h>
#include "init.h"
#include "ADXL345.h"
#include "i2c.h"
#include "terminal_comm.h"
#include "lcd.h"

// Function: ADXL345_init()
// Descrip : Initializes ADXL345
void ADXL345_init()
{
    //Clear the power control
    accelbytew(POWER_CTL,0);
    //Set ADXL to start measuring
    ADXL345_set_bit(POWER_CTL,MEASURE_E,ENABLE);
}
// Function: ADXL345_error_handle_read()
// Input   : address to read from ADXL345
// Output  : Data that was read at specified address or status of the read
// Descrip : Reads a register of the accelerometer and prints an error message
//           if it fails
unsigned char ADXL345_error_handle_read(int addr)
{
    //Read register
    int val = accelbyter(addr);
    unsigned char error_msg[] = {"Accelerometer read failed!\n\r"};
    //Check if read failed
    if(val == SEND_FAILED_CODE)
    {
        basic_error(error_msg);
        return 0;
    }
    else
        return val;
}
// Function: ADXL345_error_handle_read_m()
// Input   : address to read from ADXL345, read value 1, read value 2
// Descrip : This function does two consecutive reads to the ADXL345 and prints an error
//           message if it fails
void ADXL345_error_handle_read_m(int addr, unsigned char *read1, unsigned char *read2)
{
    //Read two values
    int val = accelbyter_m(addr,read1,read2);
    unsigned char error_msg[] = {"Accelerometer read failed!\n\r"};
    //Check if read failed
    if(val == SEND_FAILED_CODE)
    {
        basic_error(error_msg);
    }
}
// Function: ADXL345_error_handle_write()
// Input   : address in ADXL345 and data to write
// Descrip : This function does a write to the ADXL345 and prints an error message if
//           it fails
void ADXL345_error_handle_write(int addr, unsigned char databyte)
{
    //Write value to ADXL345
    int val = accelbytew(addr,databyte);
    unsigned char error_msg[] = {"Accelerometer write failed!\n\r"};
    //Check if write failed
    if(val == SEND_FAILED_CODE)
    {
        basic_error(error_msg);
    }
}
// Function: ADXL345_set_bit()
// Input   : address in ADXL345, the bit to set, and the bit value
// Descrip : This set/clears a single bit in an ADXL345 register
void ADXL345_set_bit(int addr, unsigned char bit_ofst, unsigned char bit_val)
{
    //Read current status of address to set/clear
    unsigned char status = ADXL345_error_handle_read(addr);
    //If setting bit
    //Or 1 in the bit location
    if(bit_val)
        status |= (1 << bit_ofst);
    //If clear bit
    //And the inverse of bit location into the status
```

```
84          else
85              status &= ~(1 << bit_ofst);
86          //Write value to specified address
87          ADXL345_error_handle_write(addr, status);
88
89      }
90      // Function: ADXL345_getoffset()
91      // Input    : read values to store x,y, and z axis
92      // Descrip : This function reads the offset values the accelerometer reads
93      void ADXL345_getoffset(unsigned char *x, unsigned char *y, unsigned char *z)
94      {
95          *x = ADXL345_error_handle_read(X_OFFSET);
96          *y = ADXL345_error_handle_read(Y_OFFSET);
97          *z = ADXL345_error_handle_read(Z_OFFSET);
98      }
99      // Function: ADXL345_enable_bypass()
100     // Descrip : Enables bypass functionality of accelerometer. FIFO is not used
101     void ADXL345_enable_bypass()
102     {
103         ADXL345_error_handle_write(FIFO_CTL,BYPASS_E);
104     }
105     // Function: ADXL345_disable_ints()
106     // Descrip : Disable accelerometer interrupts
107     void ADXL345_disable_ints()
108     {
109         ADXL345_error_handle_write(INT_ENABLE,DISABLE);
110     }
111     // Function: format_data()
112     // Input    : format
113     // Descrip : Sets specific format settings for ADXL345
114     void format_data(unsigned char format)
115     {
116         ADXL345_error_handle_write(DATA_FORMAT,format);
117     }
118     // Function: ADXL345_set_rate()
119     // Input    : sample rate
120     // Descrip : Sets sample rate of accelerometer
121     void ADXL345_set_rate(unsigned char rate)
122     {
123         ADXL345_error_handle_write(BW_RATE,rate);
124     }
125     // Function: ADXL345_read_x()
126     // Output   : Value read for x axis -4096 to 4096
127     // Descrip : Measures accelerometer in x direction and returns value between -4096 and 4096
128     int ADXL345_read_x()
129     {
130         //Variables
131         int x0;
132         int x1;
133         int x;
134         unsigned char x_sign;
135         //Read low and high byte of x axis measurements
136         ADXL345_error_handle_read_m(DATAX_0, &x0, &x1);
137         //Determine sign of data
138         x_sign = (x1 >> 4);
139         //Mask out sign from high byte
140         x1 = (x1 & 0x000F);
141         //Concatenate high and low byte to get 12-bit reading
142         x = ((x1 << 8 ) | x0);
143
144         //Check if the reading is negative
145         if(x_sign > 0)
146             x |= 0xFF00;
147
148         return x;
149     }
150     // Function: ADXL345_read_y()
151     // Output   : Value read for y axis -4096 to 4096
152     // Descrip : Measures accelerometer in y direction and returns value between -4096 and 4096
153     int ADXL345_read_y()
154     {
155         //Variables
156         int y0;
157         int y1;
158         int y;
159         unsigned char y_sign;
160         //Read low and high byte of y axis measurements
161         ADXL345_error_handle_read_m(DATAY_0, &y0, &y1);
162         //Determine sign of data
163         y_sign = (y1 >> 4);
164         //Mask out sign from high byte
165         y1 = (y1 & 0x000F);
166         //Concatenate high and low byte to get 12-bit reading
167         y = ((y1 << 8 ) | y0);
```

```
168         //Check if the reading is negative
169         if(y_sign > 0)
170             y |= 0xFF00;
171 
172         return y;
173     }
174     // Function: ADXL345_read_z()
175     // Output  : Value read for z axis -4096 to 4096
176     // Descrip : Measures accelerometer in z direction and returns value between -4096 and 4096
177     int ADXL345_read_z()
178     {
179         //Variables
180         int z0, z1, z;
181         unsigned char z_sign;
182         //Read low and high byte of z axis measurements
183         ADXL345_error_handle_read_m(DATAZ_0, &z0, &z1);
184         //Determine sign of data
185         z_sign = (z1 >> 4);
186         //Mask out sign from high byte
187         z1 = (z1 & 0x000F);
188         //Concatenate high and low byte to get 12-bit reading
189         z = ((z1 << 8 ) | z0);
190         //Check if the reading is negative
191         if(z_sign > 0)
192             z |= 0xFF00;
193 
194         return z;
195     }
196     // Function: ADXL345_calibrate()
197     // Descrip : Sets the x, y, and z offsets to calibrate the ADXL345
198     void ADXL345_calibrate()
199     {
200         //Variables
201         int i;
202         int avg_x = 0;
203         int avg_y = 0;
204         int avg_z = 0;
205         //Ensure ADXL345 is ready for reading
206         delay_ms(12);
207         //Read each axis 100 times
208         for(i = 0; i < 100; i++)
209         {
210             avg_x += ADXL345_read_x();
211             avg_y += ADXL345_read_y();
212             avg_z += ADXL345_read_z();
213         }
214         //Take average of each reading
215         avg_x /= 100;
216         avg_y /= 100;
217         avg_z /= 100;
218         //Calculate offset
219         avg_x = ((~avg_x) + 1)/4;
220         avg_y = ((~avg_y) + 1)/4;
221         avg_z = ((~(avg_z - 256)) + 1)/4;
222         //Write offsets to ADXL345
223         ADXL345_error_handle_write(X_OFFSET,avg_x);
224         ADXL345_error_handle_write(Y_OFFSET,avg_y);
225         ADXL345_error_handle_write(Z_OFFSET,avg_z);
226     }
227 
```

```c
// Author     : Ali Ismail
// Description: This files contais all the RS232 drivers and functions
#ifndef TERMINAL_H
#define TERMINAL_H
//defines
#define STRING_SIZE 5
#define NULL_TERM '\0'
#define BS 0x08
#define CR '\r'

//function declarations
void putchar (char c);
char getchar ();
unsigned char get_user_digit();
void clear_screen();
void basic_error(unsigned char *error_message);
#endif // TERMINAL_H

```

```c
// Author     : Ali Ismail
// Description: This files contais all the RS232 drivers and functions

#include <at89c51ed2.h>  //also includes 8052.h and 8051.h
#include <mcs51reg.h>
#include <8052.h>   // also included in at89c51ed2.h
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include "terminal_comm.h"

//Function: putchar()
//Input   : char to print to terminal
//Descrip : Writes a character to terminal
void putchar (char c)
{
    while (TI == 0);        // wait for TX ready, spin on TI
    SBUF = c;               // load serial port with transmit value
    TI = 0;                 // clear TI flag
}
//Function: getchar()
//Output  : char incoming from Terminal
//Descrip : Reads a character incoming from terminal
char getchar ()
{
//   char cc;
    while (!RI);    // wait for character to be received, spin on RI
    RI = 0;         // clear RI flag
    return SBUF;    // return character from SBUF
}

//Function: get_user_digit()
//Inputs  :
//Outputs : returns a char input from the user
//Descrip : This function returns a digit entered
//          between 1 and 6 from the user
unsigned char get_user_digit()
{
    //variables
    unsigned char read_cr;
    //grab a character from the user
    unsigned char read_char = 'a';

    while(!isdigit(read_char) || read_char < '1' || read_char > '6')
    {
        //read a char from the user
        read_char = getchar();

        //echo the character that was read
        printf("%c",read_char);

        //grab another character to see if the user wants to use this digit
        read_cr = getchar();

        //wait for a CR
        while(read_cr != CR)
        {
            // if the user backspaced then echo the result and get a new character
            if(read_cr == BS)
            {
                printf("%c",read_cr);

                read_char = getchar();
                printf("%c",read_char);
            }
            read_cr = getchar();
        }
        //let the user know if their input was invalid
        if(!isdigit(read_char) || read_char < '1' || read_char > '6')
            printf("\n\rPlease enter a valid choice 1-6.\n\r");
    }
    return read_char;

}
//Function: clear_screen()
//Descrip : Clears the display
void clear_screen()
{
    //clear the terminal screen
    //clear screen and go back to the beginning
    printf("\033[2J");
    printf("\033[0;0H");
}
//Function: clear_screen()
```

```
85  //Input   : string containing error message
86  //Descrip : Prints an error message to terminal and waits for user to hit enter
87  void basic_error(unsigned char *error_message)
88  {
89      clear_screen();
90      printf(error_message);
91      printf("Press enter to quit...\n\r");
92      //Wait until user hits enter
93      while(getchar() != CR);
94  }
95
```

```
// font.h
// Credits to http://en.radzio.dxp.pl/
// Tablica czcionek 5x7
//
//#include <avr/pgmspace.h>
#ifndef FONT_H
#define FONT_H

static const char /*PROGMEM*/  font5x8[] = {
0x00, 0x00, 0x00, 0x00, 0x00,// (space)
0x00, 0x00, 0x5F, 0x00, 0x00,// !
0x00, 0x07, 0x00, 0x07, 0x00,// "
0x14, 0x7F, 0x14, 0x7F, 0x14,// #
0x24, 0x2A, 0x7F, 0x2A, 0x12,// $
0x23, 0x13, 0x08, 0x64, 0x62,// %
0x36, 0x49, 0x55, 0x22, 0x50,// &
0x00, 0x05, 0x03, 0x00, 0x00,// '
0x00, 0x1C, 0x22, 0x41, 0x00,// (
0x00, 0x41, 0x22, 0x1C, 0x00,// )
0x08, 0x2A, 0x1C, 0x2A, 0x08,// *
0x08, 0x08, 0x3E, 0x08, 0x08,// +
0x00, 0x50, 0x30, 0x00, 0x00,// ,
0x08, 0x08, 0x08, 0x08, 0x08,// -
0x00, 0x30, 0x30, 0x00, 0x00,// .
0x20, 0x10, 0x08, 0x04, 0x02,// /
0x3E, 0x51, 0x49, 0x45, 0x3E,// 0
0x00, 0x42, 0x7F, 0x40, 0x00,// 1
0x42, 0x61, 0x51, 0x49, 0x46,// 2
0x21, 0x41, 0x45, 0x4B, 0x31,// 3
0x18, 0x14, 0x12, 0x7F, 0x10,// 4
0x27, 0x45, 0x45, 0x45, 0x39,// 5
0x3C, 0x4A, 0x49, 0x49, 0x30,// 6
0x01, 0x71, 0x09, 0x05, 0x03,// 7
0x36, 0x49, 0x49, 0x49, 0x36,// 8
0x06, 0x49, 0x49, 0x29, 0x1E,// 9
0x00, 0x36, 0x36, 0x00, 0x00,// :
0x00, 0x56, 0x36, 0x00, 0x00,// ;
0x00, 0x08, 0x14, 0x22, 0x41,// <
0x14, 0x14, 0x14, 0x14, 0x14,// =
0x41, 0x22, 0x14, 0x08, 0x00,// >
0x02, 0x01, 0x51, 0x09, 0x06,// ?
0x32, 0x49, 0x79, 0x41, 0x3E,// @
0x7E, 0x11, 0x11, 0x11, 0x7E,// A
0x7F, 0x49, 0x49, 0x49, 0x36,// B
0x3E, 0x41, 0x41, 0x41, 0x22,// C
0x7F, 0x41, 0x41, 0x22, 0x1C,// D
0x7F, 0x49, 0x49, 0x49, 0x41,// E
0x7F, 0x09, 0x09, 0x01, 0x01,// F
0x3E, 0x41, 0x41, 0x51, 0x32,// G
0x7F, 0x08, 0x08, 0x08, 0x7F,// H
0x00, 0x41, 0x7F, 0x41, 0x00,// I
0x20, 0x40, 0x41, 0x3F, 0x01,// J
0x7F, 0x08, 0x14, 0x22, 0x41,// K
0x7F, 0x40, 0x40, 0x40, 0x40,// L
0x7F, 0x02, 0x04, 0x02, 0x7F,// M
0x7F, 0x04, 0x08, 0x10, 0x7F,// N
0x3E, 0x41, 0x41, 0x41, 0x3E,// O
0x7F, 0x09, 0x09, 0x09, 0x06,// P
0x3E, 0x41, 0x51, 0x21, 0x5E,// Q
0x7F, 0x09, 0x19, 0x29, 0x46,// R
0x46, 0x49, 0x49, 0x49, 0x31,// S
0x01, 0x01, 0x7F, 0x01, 0x01,// T
0x3F, 0x40, 0x40, 0x40, 0x3F,// U
0x1F, 0x20, 0x40, 0x20, 0x1F,// V
0x7F, 0x20, 0x18, 0x20, 0x7F,// W
0x63, 0x14, 0x08, 0x14, 0x63,// X
0x03, 0x04, 0x78, 0x04, 0x03,// Y
0x61, 0x51, 0x49, 0x45, 0x43,// Z
0x00, 0x00, 0x7F, 0x41, 0x41,// [
0x02, 0x04, 0x08, 0x10, 0x20,// "\"
0x41, 0x41, 0x7F, 0x00, 0x00,// ]
0x04, 0x02, 0x01, 0x02, 0x04,// ^
0x40, 0x40, 0x40, 0x40, 0x40,// _
0x00, 0x01, 0x02, 0x04, 0x00,// `
0x20, 0x54, 0x54, 0x54, 0x78,// a
0x7F, 0x48, 0x44, 0x44, 0x38,// b
0x38, 0x44, 0x44, 0x44, 0x20,// c
0x38, 0x44, 0x44, 0x48, 0x7F,// d
0x38, 0x54, 0x54, 0x54, 0x18,// e
0x08, 0x7E, 0x09, 0x01, 0x02,// f
0x08, 0x14, 0x54, 0x54, 0x3C,// g
0x7F, 0x08, 0x04, 0x04, 0x78,// h
0x00, 0x44, 0x7D, 0x40, 0x00,// i
0x20, 0x40, 0x44, 0x3D, 0x00,// j
```

```
  0x00, 0x7F, 0x10, 0x28, 0x44,// k
  0x00, 0x41, 0x7F, 0x40, 0x00,// l
  0x7C, 0x04, 0x18, 0x04, 0x78,// m
  0x7C, 0x08, 0x04, 0x04, 0x78,// n
  0x38, 0x44, 0x44, 0x44, 0x38,// o
  0x7C, 0x14, 0x14, 0x14, 0x08,// p
  0x08, 0x14, 0x14, 0x18, 0x7C,// q
  0x7C, 0x08, 0x04, 0x04, 0x08,// r
  0x48, 0x54, 0x54, 0x54, 0x20,// s
  0x04, 0x3F, 0x44, 0x40, 0x20,// t
  0x3C, 0x40, 0x40, 0x20, 0x7C,// u
  0x1C, 0x20, 0x40, 0x20, 0x1C,// v
  0x3C, 0x40, 0x30, 0x40, 0x3C,// w
  0x44, 0x28, 0x10, 0x28, 0x44,// x
  0x0C, 0x50, 0x50, 0x50, 0x3C,// y
  0x44, 0x64, 0x54, 0x4C, 0x44,// z
  0x00, 0x08, 0x36, 0x41, 0x00,// {
  0x00, 0x00, 0x7F, 0x00, 0x00,// |
  0x00, 0x41, 0x36, 0x08, 0x00,// }
  0x08, 0x08, 0x2A, 0x1C, 0x08,// ->
  0x08, 0x1C, 0x2A, 0x08, 0x08,// <-
  0x00, 0x3a, 0x3E, 0x3a, 0x00,// heli
  0xFF, 0x00, 0x00, 0x00, 0x00,
  0x00, 0xFF,// Brick
  0x00, 0x00, 0x00, 0x00,0x00// Solid Brick
  };
  //
  #endif

```